

A MECHANISM FOR SYNCHRONOUS INTERPROCESS
COMMUNICATION OVER TRANSPARENT EXTERNAL MONITORS

Background Of The Invention

5

Field of Invention

10 The present invention relates generally to the field of
interprocess communication. More particularly, this
invention relates to an interprocess communication
mechanism that enables the synchrony semantics of each
communication to be controlled by external entities
called monitors, such that: (1) communication may be
redirected to such external entities transparently to the
sender and receiver of the communication and (2) the
monitors can determine the semantics of a completed
communication as they desire.

Background of the Invention

20 Interprocess communication (IPC) monitoring enables the
examination of any IPC between a source and a
destination. IPC monitoring is useful for a variety of
purposes, including debugging, logging, and security.
25 For example, a monitor may collect communication state
for the purpose of debugging a program consisting of
several independent tasks. Also, a monitor can be used
to filter communication data or control the communication
rate for security purposes.

30 Transparent monitoring means that the source and
destination are not aware that they are being monitored.

YOR920000575US1

This has two advantages: (1) the system can control the insertion and removal of monitors without interacting with either the source or destination, and (2) the source and destination protocols do not need to take into account the possibility that they may be monitored. Traditional systems make no attempt to support transparent monitoring. Using Mach-style ports (see K. Loepere, Mach 3 Kernel Principles, Open Software Foundation and Carnegie Mellon University, 1992), the source and destination hold rights that must be revoked in order to insert a monitor, so at least one must be notified before such a change can occur safely. The Pebble microkernel enables transparent redirection of source IPCs using its portals to implement customized IPC, but the redirection is not transparent to the destination because it sees that the message is from the redirected task, not the original (see E. Gabber et al, Building Efficient Operating Systems from User-level Components in Pebble, in Proceedings of the 1999 USENIX Annual Technical conference, 1999).

Other IPC mechanisms, such as Clans & Chiefs (see J. Liedtke, Clans & Chiefs, in Architektur von Rechensystemen, 1992, in English) and IPC Redirection (see, "Flexible Access Control Using IPC Redirection," HotOS 1999), enable monitors to intercept and forward IPCs while claiming to be the original source of the IPC. Thus, the destination receives the IPC from the source, not the monitor, so it need not know that an IPC is being monitored. Unfortunately, such mechanisms are not truly transparent because the kernel's IPC semantics are not

preserved when a monitor intercepts an IPC. Modern microkernels implement a synchronous IPC semantics, which means that the source is blocked until the destination is ready to receive the IPC or an error occurs (e.g., the destination task is killed or a timeout expires). When destination commences receipt, the IPC is sent to the destination and the source unblocks. Unfortunately, if a monitor is inserted on an IPC path, the source is unblocked when the IPC is received by the monitor, not the destination. This may result in some anomalous behaviors, such as: (1) the source assuming that IPCs have been delivered to the destination before they really have; (2) the source terminating IPCs due to timeout expiration even though the destination is ready, but because the monitor was not ready; and (3) the source assuming that the IPC was delivered reliably to the destination when an error may have occurred.

What is needed is that the system (i.e., the kernel and the monitors) control the synchrony of each communication, so that the communication appears to the sender and destination to be implemented according to the same semantics regardless of whether a monitor is present or not. Further, the system may choose arbitrary actions upon a communication, so the interprocess communication mechanism must permit the system to provide any desired semantics. Lastly, the interprocess communication mechanism must result in a system that is robust in the presence of errors.

Summary of the Invention

An object of this invention is to improve interprocess communications.

Another object of the present invention is to provide a method and system for controlling the synchrony of interprocess communications so that the communications appear to the senders and destinations to be implemented according to the same semantics regardless of whether the communications are monitored or not.

A further object of this invention is to use monitors, for monitoring interprocess communications, as extensions of a system kernel, to that, when a particular communication is monitored, the source and destination of the communication are treated as if the kernel is still processing the communication.

These and other objects are attained with a method and system for performing interprocess communications (IPCs). The method comprises the steps of receiving IPC requests, where each of the IPC requests identifies a source and a destination; building IPCs in response to the request; transmitting the IPCs from the sources to the destinations; and intercepting and examining selected ones of the IPCs. The method comprises the further step of controlling the synchrony of the IPCs so that each IPC appears to its source and destination to be implemented according to the same semantics regardless of whether the IPC is intercepted and examined.

With the preferred embodiment of this invention, the system monitors are considered as an extension of the system kernel (although they may be linked into the kernel and run in kernel mode as well), so the source and destination are treated as if the kernel is still processing the IPC. Thus, the desired semantics of communication can be implemented in the monitors. However, there are a number of possible monitoring semantics that may be reasonable, so in order to maintain utility of the approach, the IPC mechanism enables the monitors to coordinate using a small set of actions.

These actions are as follows. First, the interception of an IPC by a monitor permits the monitor to manage: (1) the identity of the source that ultimately must be unblocked; (2) when the sender is unblocked; and (3) whether the monitor is notified when the communication reaches the destination. In addition, error handling semantics can be expressed by the monitors such that all communication state in the monitors is maintained consistently.

Further benefits and advantages of the invention will become apparent from a consideration of the following detailed description, given with reference to the accompanying drawings, which specify and show preferred embodiments of the invention.

Brief Description of the Drawings

Figure 1 displays the general mechanism for delivering IPCs synchronously regardless of whether one or more

monitors are inserted on the IPC path between the source and destination.

5 Figure 2 displays how the delivery mechanism for IPCs is chosen.

Figure 3 displays how the kernel implements a source to monitor IPC delivery.

10 Figure 4 displays how the kernel implements a monitor to monitor IPC delivery.

Figure 5 displays how the kernel implements a monitor to destination IPC delivery.

15 Figure 6 displays the process by which a monitor handles IPCs that are redirected to it while managing the synchronicity information.

20 Figure 7 displays how the monitor prepares an IPC that it is forwarding.

Figure 8 displays how an IPC error message is processed.

25 Figure 9 displays how an IPC unblock message is processed.

Figure 10 displays how the synchrony is implemented when the IPC is delivered to the destination.

30 Figure 11 displays how an IPC error message is built.

Figure 12 displays how an IPC error message is delivered.

Figure 13 displays how an IPC unblock message is built.

5 Figure 14 displays how an IPC unblock message is delivered.

Figure 15 displays how an IPC notify message is built.

10 Figure 16 displays how an IPC notify message is delivered.

Figure 17 displays how the destination to which a source is blocked is changed should a monitor redirect the message to an alternative final destination.

Detailed Description of the Preferred Embodiment

For the preferred embodiment, the L4 microkernel's IPC mechanism as the base IPC mechanism. The L4 IPC mechanism is described in the L4 Reference Manual. L4 enables the creation of tasks that can be inserted on arbitrary IPC paths. A mechanism called IPC redirection is used in the preferred embodiment to determine whether an IPC is sent to a monitor or to a destination ("Flexible Access Control Using IPC Redirection," HotOS 1999). One suitable redirection mechanism is disclosed in copending patent application no. 09/276,869, filed March 26, 1999, entitled "Flexible Interprocess Communication via Redirection," the disclosure of which is herein incorporated by reference.

Figure 1 shows the basic synchronous IPC mechanism that enables transparent use of an arbitrary number of monitors. First, the source requests an IPC send (100). The sender (in this case, the source) is then blocked (110). The kernel processes the IPC delivery (200) which results in the IPC being sent either to the destination or a monitor (120). The kernel uses the IPC redirection mechanism to determine whether the IPC is delivered to the destination or a monitor. If the IPC is delivered to the final destination, then the kernel completes the synchronous IPC processing to ensure that the source can be unblocked (1000). If the IPC is delivered to a monitor, then the monitor performs its processing and maintains the synchrony information of the IPC (700). The monitor then decides whether to forward the IPC to the destination (or an alternative destination) or not (130). If the IPC is forwarded, the monitor sends the IPC using the mechanisms described in (200). Otherwise, the monitor decides whether to return an error (140). Error messages are a new type of IPC, so a new mechanism is described for handling them (800). If no error is returned, the monitor simply waits for a subsequent request (150).

Figure 2 describes how the IPC delivery mechanism is chosen (200). This mechanism is executed by the kernel in the preferred embodiment. First, the IPC redirection mechanism determines the immediate destination of an IPC (210) (220): (1) an invalid destination indicates that the IPC is to be blocked (230); (2) the IPC may be sent directly to a destination; or (3) the IPC may be redirected to a monitor. In addition to the L4 IPC

redirection mechanism, monitor tasks must be signified to the kernel, so completion processing can proceed (1000). In either case, the IPC may be deceived (i.e., sent from a task claiming to be another (240), and the delivery mechanism choice depends on whether a monitor or the source directly sent the IPC (250). Different mechanisms are applied for: (1) source to destination delivery (260) (this is the standard L4 IPC delivery mechanism, so it is not covered); (2) source to monitor delivery (300); monitor to monitor delivery (400); or monitor to destination delivery (500).

Figure 3 describes how the kernel processes an IPC delivery from a source to a monitor (300). The kernel simply delivers the IPC from the source to the monitor (310) and unblocks the monitor (320). This delivery (310) is performed using the traditional deceiting IPC as described in the L4 Users' Manual. After the IPC is delivered, only the monitor is unblocked (320). The source remains blocked.

Figure 4 describes how the kernel processes an IPC delivery from a monitor to another monitor (400). First, the kernel sets the traditional deceiting IPC data for the IPC as described in the L4 Users' Manual (410). Then, the kernel determines the type of the IPC (420) and sets the IPC type value appropriately (430). The various IPC type values are listed below:

when the IPC has been completed (i.e., been delivered to the destination); or (3) the current controlling monitor value it received in the incoming IPC. Subsequently, the monitor completes and delivers a typical L4 deceived IPC (766) and then waits for further redirected IPCs (150).

Figure 8 describes how an IPC error message is processed (800). First, the monitor builds an IPC error message (810). Second, the monitor asks the kernel to deliver the IPC error message using the basic L4 IPC mechanism (820). The IPC error message is itself an IPC, so it can be delivered using a slight generalization of the basic IPC mechanism. Then, the monitor waits for the next IPC (150).

Figure 9 describes how an IPC unblock message is processed (900). First, the monitor builds an IPC unblock message (910). Next, the monitor asks the kernel to deliver the IPC unblock message (920). Like the IPC error message case, the IPC unblock message is also an IPC. Lastly, the monitor waits for the next IPC (150).

Figure 10 describes the additional IPC effort when the IPC is delivered to the destination from the monitor (1000). At this time, the kernel must determine how to proceed with unblocking the source. If a controlling monitor is set (1010) (1020), then a notify message is constructed which is used to inform the controlling monitor that IPC from the source has been completed (1030). This message is delivered to the controlling monitor (1040). If no controlling monitor is set, the kernel then determines whether a blocked source is set

(1050) (1060). If so, the blocked source task is unblocked (1070). Otherwise, the value of the true source is unblocked (1070).

5 Figure 11 describes the mechanism for building error messages (810). The key component is the setting of the IPC error type (812). Following this, the error code of the IPC is set (814). Any IPC message for the source is added (816). Lastly, the blocked source of the IPC is set, so the IPC error is delivered to the proper task (818).

10 Figure 12 describes how IPC error messages are delivered by the kernel (820). The kernel first determines if the monitor is on the IPC path source and destination (821). If so (822), the kernel determines if the source is blocked on the destination (823). If the source is blocked on the destination (824), then the kernel sets the IPC error state for the source (825) and unblocks the source (826). If the monitor is not on the proper IPC path (822) or the source is not blocked on the destination (824), the IPC error message is terminated and an error is returned to the monitor (410).

20 Figure 13 describes the mechanism for building unblock messages (910). The key step is the setting of the IPC unblock type (911). Following this, the error code of the IPC is set (912). Any IPC message for the source is added (816). Lastly, the blocked source of the IPC is set, so the IPC unblock is delivered to the proper task (818).

Figure 14 describes how IPC unblock messages are delivered by the kernel (920). The kernel first sets the blocked source identity to that of the source in the IPC (921). Next, the kernel determines if the monitor is on the IPC path between the blocked source and the current destination (821). If so (822), the kernel determines if the source is blocked on the destination (823). If the source is blocked on the destination (824), then the kernel sets the IPC error state for the source to that specified in the unblock IPC (825) and unblocks the blocked source (1070). If the monitor is not on the proper IPC path (822) or the sender is not blocked on the destination (824), the IPC error message is terminated and an error is returned to the monitor (410).

Figure 15 describes how a IPC notify message is built that informs a controlling monitor of a completed IPC. First, the kernel obtains the blocked source of the active IPC (1050). If there is a blocked source (1060), the kernel sets the true source of the notify IPC to the blocked source (1032). If there is no blocked source (1060), then the true source of the notify IPC is set to the true source of the active IPC (1033). The kernel sets the destination in the notify IPC to the controlling monitor in the active IPC (1034). Lastly, the kernel sets the IPC type to notify (1035).

Figure 16 describes how the kernel delivers a notify IPC. The kernel first verifies that the controlling monitor of the active IPC is on the IPC path between the notify IPC's true source and the active IPC's destination (1041). If it is on the IPC path (822), then the kernel

delivers the notify IPC as a typical IPC (200).
Otherwise, the monitor receives a notify error from the
kernel in its error state (1042).

5 Figure 17 describes how the kernel updates the blocked
source's destination when the monitor specifies a change
in destination (770). First, the kernel gets the blocked
source value (1050). If there is a blocked source
(1060), then the source of the changed destination is
10 then set to the blocked source (771). Otherwise, the
source of the changed destination is set to the true
source (772). Next, the new destination is retrieved
(823). If the source is blocked (824), then the
destination in which it is blocked is changed to the new
destination (775). Otherwise, no action is taken (774).

While it is apparent that the invention herein disclosed
is well calculated to fulfill the objects stated above,
it will be appreciated that numerous modification and
embodiments may be devised by those skilled in the art,
and it is intended that the appended claims cover all
such modifications and embodiments as fall within the
true spirit and scope of the present invention.